CSE 403
Lecture 23

Complexity Theory and Software Engineering

Announcement

- Project submission
- Project demos (Wednesday)
- Exam review (Monday)
- Exam format
  - Short answer

Theory of Computation

- What can complexity theory tell us about software engineering?

Today's result

- Syntactic measures of program complexity
  - Thm:
- Automatic evaluation of program correctness
  - Thm:
- Evaluation of finite state systems
  - Thm:

Software metrics

- Is there a meaningful way of evaluating the quality of a program.
- We would like to have ways to look at source code, and evaluate "simplicity" or "risk of bugs"

Straw men

- The fewer loops the better
- Unnested loops are better than nested loops

Matrix multiplication

```
MatrixMult(A, B, C){        // A x B = C
   for (int i = 0; i < n; i++)
     for (int j = 0; j < n; j++){
        int t = 0;
        for (int k = 0; k < n; k++)
            t += A[i][k]*B[k][j];
        C[i][j] = t;
     }
 }
```

Reducing complexity

- Can the level of nesting by reduced?
- Can this be implemented with a single loop?
- How about with no loop (straight line code)
- Function calls not allowed

Loop removal theorem

- Any program can be rewritten to have just a single loop and no function calls
- First complexity theory ideas
  - Convert a program to a simple intermediate language
  - Write an interpreter for the program that only has a single loop

Interpreter

```
while (pc != stop){
     if (pc == 1)
          Execute statement 1
     if (pc == 2)
          Execute statement 2
     . . .
}
```

Interpretation of the result

- Thm: Any program can be converted to a single loop program
- Simplicity is not related to depth of loops, number of loops, number of function calls
- Important properties of programs are semantic, not syntactic

The halting problem

- It is impossible to write a program which can always determine whether or not an input program halts
- Philosophical result – limits on power of computation

Halting Problem

- Suppose we have a program Halt(P, y) which return true if P halts on input y and returns false otherwise

- Define

  Halt'(P) = if Halt(P, P) then loop else return

  Q is the program for Halt'(P)

Does Q(Q) Halt?

Q(P) runs forever if P halts on input P

Q(P) halts if P runs forever on input P

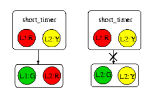
If Q(Q) halts, then Q(Q) runs forever

If Q(Q) runs forever, then Q(Q) halts

So what????

- Testing software is undecidable
  - If we can't test does it halt, we can't test if it computes a particular value
  - Testing if a certain line of code is reachable is undecidable (because the line could be the return statement)
  - Testing if a variable is always initialized is undecidable (because we might have a statement that can reach it without initialization, and then test to see if that statement is reachable)

Finite state systems



- If things are finite, they become much easier
- Reachability in finite automata
- Equivalence of finite automata
- Evaluation of temporal formulas for finite state systems
  - IF A THEN EVENTUALLY B
  - IF (A FOLLOWS B) THEN (ALWAYS C UNTIL D)

Pseudo Finite Systems

- However, finite can be very large
- The state space can be much larger than the system description
- Add algorithms can be very inefficient
  - Double Exponential, Triple Exponential
- There is a large amount of interesting research in
  - Representing infinite spaces by finite spaces
  - Applying finite state methods to software